Emre Zeytinoğlu 28190

CS307-PA4

Since it was mentioned in the assignment, I used a HeapManager class and HeapNode configuration to set up a simple memory management system in the code. The purpose of this system is to manage the memory areas needed by the program and control memory allocation processes.

HeapNode Structure

The HeapNode structure forms the memory management system. Each node stores the properties of the memory block mentioned in the assignment. Additionally, I added a "next" section to create a linked list and to navigate within the linked list easily:

- id: The identification number of the memory block.
- size: The size of the block (in bytes).
- index: The starting position or index in memory.
- next: Points to the next HeapNode.

HeapManager Class

This class contains a set of methods to perform memory management operations. I created all the functions mentioned in the assignment in this class:

Constructor (HeapManager()): The initializer sets the head pointer to NULL.

Destructor (~HeapManager()): Destructor clears all created HeapNodes.

initHeap(int size): Using this function, we can initially create a single HeapNode that represents the entire memory block.

myMalloc(int ID, int size): Allows memory allocation of a certain size. If there is enough space, we can divide it into the desired dimensions by creating a new HeapNode.

myFree(int ID, int index): Frees the memory block with the specified ID and index. Then, it merges adjacent free blocks.

print(): Shows the status of current memory blocks. The ID, size and index of each block are printed on the screen.

Synchronization

The code maintains data integrity under multi-threading conditions by using a mutex (mutual exclusion) named lock of type pthread_mutex_t. This prevents multiple threads from performing memory management operations simultaneously, preventing potential conflicts and data corruption.

Pseudocodes For Locking Algorithm

myMalloc

```
Start of myMalloc function
   Lock with lockMalloc
   Attempt to allocate memory
   If successful
      Allocate memory and release the lock
      Return the index of allocated memory
   If unsuccessful
      Release the lock and return an error
```

myFree

```
Start of myFree function
   Lock with lockFree
   Attempt to free memory
   If successful
      Free memory and release the lock
      Return success status
   If unsuccessful
      Release the lock and return an error
```

print

```
Start of print function
   Lock with lockPrint
   Print memory status
   Release the lock
```